futuredata

MICROKIT EXTENDED BASIC

REFERENCE MANUAL

MICROCOMPUTER
SYSTEMS

Received 11/14/78

MICROKIT EXTENDED BASIC

REFERENCE MANUAL

MICROKIT INC
11205 S. La Cienega Blvd.
Los Angeles, CA 90045
(213) 641-7700

Copyright 1977, MICROKIT INC

MICROKIT EXTENDED BASIC

REFERENCE MANUAL

MICROKIT INC.
11205 S. La Cienega Blvd.
Los Angeles, CA 90045
(213) 641-7700

Copyright 1977, MICROKIT INC.

TABLE OF CONTENTS

	Page
1.0 Introduction	1-1
1.1 Loading BASIC	1-2
1.1.1 Loading from Tape	1-2
1.1.2 Loading from Diskette	1-2
1.2 BASIC Editor	1-3
1.2.1 Tape BASIC Editor	1-3
1.2.2 Disk BASIC Editor	1-3
1.3 BASIC Monitor	1-5
2.0 Statement Format	2-1
3.0 Data Types	3-1
3.1 Precision	3-1
3.2 Real Numbers	3-1
3.3 Integer Numbers	3-2
3.4 Strings	3-3
3.5 Arrays	3-4
4.0 Statements	4-1
CONT	4-1
DATA	4-1
DEF	4-1
DIM	4-2
EDIT	4-2
END	4-2
EXEC	4-2
FLOW	4-3
FOR	4-3
GOSUB	4-3
GOTO	4-3
IF	4-3
INPUT	4-4
LET	4-4
MEMEND	4-4
NEXT	4-4
NOFLOW	4-5
ON ... GOSUB	4-5
ON ... GOTO	4-5
OUT	4-5
POKE	4-5
PRINT	4-6
SIZES	4-6
STOP	4-7

Page

STRING 4-7
RANDOMIZE 4-7
READ 4-7
REM 4-7
RESTORE 4-8
RETURN 4-8
RUN 4-8

5.0 Operators 5-1

6.0 Functions 6-1

6.1 String Functions 6-1

CHAR 6-1
ASC 6-1
LEFT 6-1
RIGHT 6-1
MID 6-1
LEN 6-1
REPEAT 6-2
INDEX 6-2
VERIFY 6-2
STR 6-2
FM 6-2
VAL 6-2
MIN 6-2
MAX 6-2

6.2 Numeric Functions 6-3

INT 6-3
FIX 6-3
FRAC 6-3
ABS 6-3
SGN 6-3
MIN 6-3
MAX 6-3
MOD 6-3
RND 6-4
SQR 6-4
LOG 6-4
LN 6-4
EXP 6-4
SIN 6-4
COS 6-4
TAN 6-4
ATN 6-4

6.3 Other Functions 6-5

CLOCK 6-5
IN 6-5

Page

PEEK 6-5
PGMSIZE 6-5
SPACELEFT 6-5

6.4 FM (Format) Function 6-6

6.4.1 FM Picture Format Characters 6-7
6.4.2 FM Examples 6-8

7.0 File Input/Output 7-1

7.1 File I/O Features 7-1

7.1.1 Record Based I/O 7-1
7.1.2 Sequential and Random I/O 7-2
7.1.3 Editor Compatible Files 7-2
7.1.4 End-Page, End-File and Error Exits 7-3
7.1.5 Record Length Specification 7-3
7.1.6 Data I/O Format 7-3
7.1.7 I/O to Disk 7-4

7.2 File I/O Statements 7-5

7.2.1 OPEN Statement 7-5
7.2.1.1 OPEN Device Specification 7-5
7.2.1.2 OPEN Parameters 7-6

EDIT 7-6
CLEAR 7-6
END 7-6
ERROR 7-7
ENDPAGE 7-7
PAGESIZE 7-7
LENGTH 7-7
XRATE 7-8
NULLS 7-8

7.2.2 Other File I/O Statements 7-9

CLOSE 7-9
GET 7-9
PUT 7-10
EJECT 7-11
DELETE 7-11
FREESPACE 7-11
RENAME 7-11
ATTRS 7-12
EOF 7-12
GETSEEK 7-12
PUTSEEK 7-12
LOAD 7-13
REWIND 7-13

Page

7.3 File I/O Functions 7-14

NAME 7-14
ATTR 7-14
TRACKS 7-14
FREETRACKS 7-14
SIZE 7-14
RECGET 7-14
RECPUT 7-14
ERR 7-15
ERR$ 7-15

8.0 Assembly Language Subroutines 8-1

9.0 Sample BASIC Program 9-1

MICROKIT EXTENDED BASIC

1.0 INTRODUCTION

MICROKIT BASIC is made up of two main elements: (1) the BASIC Monitor, and (2) the BASIC Editor.

The BASIC Editor is nearly identical to the standard MICROKIT Editor and is used to edit BASIC programs. It is also used to load and store programs on cassette tape or diskette. Editing is performed as a separate function from program execution, which is handled by the BASIC Monitor.

The BASIC Monitor provides the capability for executing a program loaded into the system memory and residing in the BASIC Editor work area. BASIC statements can be executed by the Monitor directly, or execution can be initiated using the RUN command. Considerable debugging power is provided by the capability to interrupt execution of a program and display the values of execution variables. When program changes must be made the Editor is invoked to perform the modifications.

Since BASIC programs are stored on cassette tape or diskette in a form compatible with the MICROKIT Editor, it is also possible to edit programs using the regular Editor.

MICROKIT Extended BASIC provides numerous special features and capabilities. Because many programmers are interested in maintaining some degree of compatibility with other BASIC systems, all MICROKIT extended features are marked with two asterisks, "**".

This manual contains reference material for the BASIC monitor and interpreter. For further information on the Editor, refer to the "MICROKIT Editor Reference Manual". For further information on operating the system and on using the cassette tape units, refer to the "MICROKIT Operating and Programming Manual". And for further information on using the disk system, the disk-based Editor, and on the disk data file structure, refer to the "MICROKIT Disk Operating System Reference Manual".

1.1 Loading the BASIC System

For completeness, this section is included to summarize the loading process. More information on loading from cassette tape is provided by the "MICROKIT Operating and Programming Manual," while more information on loading from diskette is provided by the "MICROKIT Disk Operating System Reference Manual."

1.1.1 Loading from Tape

Tape-based BASIC is loaded by placing the BASIC System Tape in cassette unit 1, pressing the PLAY key, and initiating the bootstrap loader using the LOAD key on the system keyboard. The tape unit should begin running, and as the system reads the tape, the "Enabled" LED on the keyboard will blink if all is going well. On systems where bootstrap loading from disk is also possible, the bootstrap will require that a "T" key be typed to specify loading from tape.

When loading is complete, a REWIND message appears on the CRT display and the tape should be rewound by pushing the REW key on the tape unit. When this has been completed, the RETURN key on the keyboard signals the system that the rewind is completed, and BASIC receives control.

1.1.2 Loading from Diskette

Disk-based BASIC is loaded by inserting the BASIC System Diskette in disk drive 0, closing the drive door, and initiating the disk bootstrap loader using the LOAD key on the keyboard. After the bootstrap is initiated, a "D" key must be typed to select loading from diskette. Data transfer from diskette begins and in several seconds the Disk System Monitor is in control.

BASIC is reached from the system monitor by typing "J B" followed by a RETURN.

1.2 BASIC Editor

The BASIC Editor receives control when the BASIC system is first loaded. This gives the programmer an opportunity to enter a program from the keyboard or load a previously saved program from cassette tape or diskette.

The BASIC Editor is nearly identical to the usual MICROKIT Editor with several additions. The tape based version of BASIC incorporates the tape version of the MICROKIT Editor, while the disk based version includes the disk Editor.

A "Q" (Quit) command has been added to both Editors. This provides for transfer from the BASIC Editor to the BASIC Monitor so that execution can be initiated. The default tab stops in the Editor have been set at convenient locations for BASIC programming.

Both the BASIC Editor and the BASIC Monitor remain resident in memory during use of the BASIC system. This makes the transfer from editing mode to execution/debugging mode rapid and convenient. The BASIC program remains in the Editor work area at all times during the use of the BASIC system, and is only erased or replaced using Editor commands.

1.2.1 Tape BASIC Editor

When using the tape based version of BASIC, programs can be loaded from a tape in cassette unit 1 using the "L" command. Once the program is loaded, the tape must be rewound using the "REW" command.

Programs are saved on cassette tape unit 2 using the "E" command. This command causes the program held in the Editor work area to be written out to tape, but retains the program in the work area for additional modifications or execution. The program can be erased from memory using the "CLR" command.

1.2.2 Disk BASIC Editor

BASIC programs are loaded in the disk version using the "L filename" command. Programs should be saved on diskette with the "W* filename" command. This command causes the program to be written to disk but retains the program in memory for further modification or execution.

Transfers from BASIC to other Disk System functions are accomplished using the "J" command. This command is recognized

only in the BASIC Editor, so that entry into and exit from BASIC takes place through the Editor. This eliminates the possibility of transferring out of BASIC before the program is saved.

Other disk management commands are provided by the MICROKIT Disk Operating System Monitor. For further information about the disk Editor and disk data management refer to the "MICROKIT Editor Reference Manual" and the "MICROKIT Disk Operating System Reference Manual".

1.3 BASIC Monitor

The BASIC Monitor provides the capability of interpretively executing BASIC statements. When the Monitor receives control, it displays the message "READY". At this point, the user has several choices.

Entering a BASIC statement followed by a RETURN key, will cause the statement to be executed immediately. This feature can be used to print values, change values of variables, or "GOTO" specific statements within a program.

Typing the "RUN" command causes the BASIC Monitor to begin executing the statements stored in the Editor work area. When execution is initiated with the RUN command, all variables are initialized to zero. The BASIC program executes until an interrupt occurs or until a STOP statement, END statement or the end of the program is reached. The user can interrupt a BASIC program by pressing the BRK key. BASIC programs can be continued after being stopped by using the CONT command.

Transferring from the BASIC Monitor to the BASIC Editor is accomplished using the "EDIT" command. When this command is invoked, control is returned to the Editor where the program can be displayed, modified, or saved. The Editor display will be positioned at the program statement last executed by the BASIC monitor.

When a programming error is detected during execution, BASIC stops and displays an informative message, along with the the BASIC statement in which the problem was detected. If the problem is not immediately obvious, additional debugging information can be obtained by using the PRINT statement to directly display the values of variables in the statement.

When BASIC program execution is suspended by the detection of an error, by an interrupt using the BRK key, or by a STOP or END statement, the EDIT command can be used to transfer to the Editor for display of the program. If changes are made to the program while in the Editor it may not be possible for execution to resume.

When control is returned to the BASIC monitor from the Editor, a message is displayed to indicate whether program execution can or cannot continue. The messages are as follows:

VARIABLES RETAINED

> Indicates that the values of all variables set prior to entering the Editor have been retained during the

editing process.

If the Editor is entered to view the program during suspended execution, and if no changes are made, the programmer may exit from the Editor and resume execution where it was interrupted.

If changes are made to the program, execution cannot be resumed using the CONT command. All variable values are unchanged, however, and may be displayed using the PRINT statement. Also, it may be possible to resume execution using the GOTO statement to start at a particular statement within the program.

VARIABLES ERASED

Indicates that significant changes were made to the program, and the variable area was affected. Therefore, no variable values are available for display, and execution should be restarted from the beginning using the RUN command.

2.0 STATEMENT FORMAT

Typically each line has a statement number and one BASIC statement. In MICROKIT BASIC, however, statement numbers are optional, and need not be in ascending sequence. Statement numbers must be provided on statements that are referenced by branching statements such as GOTO, GOSUB, or IF ... THEN.

When statement numbers are used, they must be one to four digits in length and must begin at the left-most position in the line. When the statement number is omitted, the statement may begin at any position in the line.

MICROKIT BASIC statements can be extened to multiple lines by placing a hyphen, "-", in column one of each continued line. However, keywords and constants cannot be broken across line boundaries. Multiple statements can be entered on a single line by separating the statements with a colon, ":". A statement number on a line with multiple statements refers only to the first statement in the line. Furthermore, a statement number can be used only with the first statement on a line.

3.0 DATA TYPES

BASIC provides three data types: Reals (floating point), Integers (fixed point) and Strings. Automatic conversions are performed from integers to reals when they are mixed in a single statement. Conversions between numbers and strings must be explicitly expressed using the STR, VAL, FM, CHAR or ASC functions.

3.1 Arithmetic Precision

A unique and powerful feature of MICROKIT BASIC is the ability to perform extended precision arithmetic. Real numbers and integer numbers can be specified to have up to 200 digits of precision. The default precision is 8 digits for real numbers and 6 digits for integer numbers.

If the precision is changed from the default, it must be done prior to the execution of a BASIC program, i.e. as the first statement of a program, and may not be changed again during the execution of the program. The changing of the default precision affects all real or integer numbers, and cannot be done on a selective basis.

The size of real and integer numbers is specifed as the number of memory bytes used for the storage of the respective values, and is changed using the SIZES statement. For each data type in the sections that follow, the formula for determining the number of digits of precision from the size in bytes is given.

MICROKIT BASIC provides primary arithmetic functions on reals and integers to the full precision specified. These functions include add, subtract, multiply, and divide. Higher level math functions are provided and have a maximum accuracy of 21 digits. These functions include SIN, COS, ATN, LOG, EXP, etc.

The following sections define the characteristics of the BASIC data types, including real numbers, integer numbers, strings, and arrays. Constants, variabls, range, and internal format is given for each data type.

3.2 Real Numbers

a) Real Constants:

Any sequence of digits with an optional sign, decimal point and exponent.

Examples:

```
123.45
-12.34
123.45E10
-12.345E-4
```

b) Real Variables:

Any single letter or any single letter followed by a digit.

Examples:

```
A B2 C Z9
```

c) Range of Real Numbers:

The range of a real number is 1x10↑-63 to (1x10↑64)-1 and the precision is 2(RSIZE-1) decimal digits, where RSIZE is the number of memory bytes specified for real numbers. If this value is not changed by executing a SIZES statement, its default value is 5, which provides 8 digits of precision.

d) Internal Format of Real Numbers:

Byte 1: 1 bit sign and 7 bit exponent excess 64.
Bytes 2 to RSIZE: each byte stores 2 BCD digits.

3.3 Integer Numbers

a) Integer Constants:

A sequence of digits with an optional sign, but without a decimal point or exponent.

Examples:

```
1234
-23
```

Constants can also be specified in binary, octal and hexadecimal (actually any base between 2 and 16) as follows: one or two digits specifying a radix between 2 and 16, followed by the character "R", followed by a number expressed with the digits 0 to 9 and the letters A to F.

Examples:

```
16R9F - "9F" in hexadecimal
10R67 - "67" in decimal
8R470 - "470" in octal
2R101 - "101" in binary
```

b) Integer Variables:

Any single letter followed by the character "%".

Examples:

```
A% B% Z%
```

c) Range of Integer Numbers:

The range of an integer number is between -0.5x10↑(2*ISIZE)+1 and 0.5x10↑(2*ISIZE), where ISIZE is the number of bytes specifed for integer numbers. If not changed by the SIZES statement, the default specification is 3, which provides for integers ranging between 500,000 and -499,999.

d) Internal Format of Integer Numbers:

Byte 1 to ISIZE: 2 BCD digits stored in ten's complement.

3.4 Strings

a) String Constants:

A sequence of characters delimited by quotes, e.g. "A STRING". If quotes are used, they must be doubled, e.g. """A""" is the three character string representing "A".

b) String Variables:

Any single letter followed by the character dollar, "$".

Examples:

```
A$ C$ Z$
```

c) Size of Strings:

The memory space reserved for a string variable is either declared using the DIM statement or is the default reservation of 40 characters. This automatic default can be changed using the SIZES statement. The maximum value

that can be specified as a space reservation is 250.

Strings are varying length sequences of characters up to a maximum length equal to the number of bytes reserved for the string. All strings have an associated length value that specifies the length of the string stored in the string variable. The actual length of string variables may be less than the maximum size permitted by the space reserved for the variable, and when a value is assigned to a string variable, the length of the new value updates the current length of the variable. If the string to be assigned is longer that the maximum length that can be stored in a particular variable, the string is truncated on the right.

d) Internal Format of Strings:

Byte 1: Maximum Length
Byte 2: Current Length
Byte 3 to byte N: Any ASCII character (where N=2+Maximum Length found in byte 1)

3.5 Arrays

a) Array Constants:

None.

b) Array Variables:

Real arrays are denoted with a single letter, integer arrays are denoted with a single letter followed by a percent character, "%", and string arrays are denoted with a single letter followed by the dollar character, "$".

Array variables are similar to non-array variables, except that when referenced, they must always be followed by a parenthesized list specifying the array element.

c) Size of Arrays:

Arrays are zero indexed and can have one to four dimensions. All arrays must be explicitly declared using the DIM statement.

4.0 STATEMENTS

This section contains a complete description of each of the statements of MICROKIT Extended BASIC, with the exception of File I/O statements described in a later section.

CONT

Used to continue execution of a program from the BASIC Monitor. After any STOP statement or user interrupt a BASIC program can be continued from where it left off. Alternatively, RETURN may be used instead of CONT since internally a GOSUB is generated for each STOP or interrupt.

DATA constant list

```
DATA 2,-3,"JIM"
```

Specifies data to be used with the READ and RESTORE statements. A constant list is a sequence of number or string constants separated by optional commas.

DEF FNa(variable) = expression

```
DEF FNa = expression
DEF FNK(A$) = "<" + A$ + ">"
DEF FNP = 3.14159*R↑2
```

Allows user defined functions to be declared. One or zero parameters of any type is allowed. The name of a function is the letters "FN" followed by a third letter.

**DEF FAa = expression

```
DEF FAA = 16R6100
```

Defines the entry addess of user defined assembly language routines. The function name must "FA" followed by a letter. When the function is used in a program, BASIC transfers control to the specified memory address. It is assumed that the programmer has loaded his subroutine at this address. If the assembly language subroutine contains errors, or if an incorrect address is specified, the system response may be unpredictable. Refer to the section on user defined assembly functions for further information.

DIM array list with dimensions
DIM A(5,5),B$(100,1),C$(200), D(F+G)

This statement is used to specify the size of array variables. An array can have up to four dimensions. **Dimension statements are executed dynamically so that dimension bounds can be expressions. The last dimension in a string array is not a dimension, but rather declares the maximum length of each element in the array. If only one dimension appears for a string variable then it is not an array but simply a string variable with a declared maximum length. In the example above, B$ is a 100 element one dimensional array of character strings with a maximum length of one character. C$ is not an array, but instead a string variable with a maximum length of 200 characters.

**EDIT

Used to transfer to the BASIC Editor. The Editor provides for modifying BASIC programs and for the loading and storing of BASIC programs. When the Editor is entered after execution is suspended, the display will contain the statement which was last executed. This line may be a STOP statement, a statement with an error or the statement which the user interrupted. The EDIT statement is only an interactive command and may not appear within a BASIC program.

END

The last statement in a BASIC program. **The END statement is optional and may be placed anywhere to end the program.

**EXEC expression
EXEC A$
EXEC "PRINT 2"

The EXEC statement evaluates and then executes the string expression which may consist of one or more BASIC statements separated by colons. This may be used for evaluating expressions or statements entered as INPUT data.

**FLOW

Turns on the flow trace. When the flow trace is enabled by executing the FLOW statement, all statements are displayed as they are executed. This allow the user to watch the flow of the program execution during debugging. The NOFLOW statement is used to disable the flow trace.

```
FOR variable = expression TO expression [STEP expression]
FOR J=1 TO 10
FOR K=20 TO 2 STEP -2
```

This is used for repeated execution of a group of statements between the FOR and corresponding NEXT statement with the loop variable stepping from the starting value to the ending value. STEP is the amount by which the loop variable is changed on each iteration. STEP is optional and if omitted is assumed to be one.

```
GOSUB statement number
GOSUB 600
```

Transfers execution to the statement number specified. Execution continues in the "subroutine" until a RETURN statement is encountered at which point execution is transferred back to the statement following the GOSUB.

```
GOTO statement number
GOTO 600
```

Transfers control to the specified statement number.

```
IF expression THEN statement number
IF J=4 THEN 200
**IF expression THEN statement
IF K>L+1 THEN L=L+1:C$="YES"
```

This is used for conditional branching or conditional execution of a statement. The expression is evaluated as a logical expression. If the expression evaluates to true, the THEN statement is executed, otherwise the THEN statement is skipped, and execution continues with the next statement.

If a statement number or expression which evaluates to a statement number appears to the right of the THEN, and if the test expression evaluates true, execution is transferred to the specified statement number. If the test expression does not evaluate true, execution continues with the next statement.

If there is a statement or series of statements (separated by colons and on the same possibly continued line) to the right of the THEN, and if the expression to be tested evaluates true, the statement(s) will be executed. If the expression is not true, the statement(s) will be skipped.

INPUT variable list
INPUT M,N,A$

Reads values from the keyboard and assigns those values to the corresponding variable. The data is a sequence of numeric or string constants separated by blanks and optional commas. If strings are not enclosed in quotes, then the end of the string is either the end of the data typed, or is delimited by the character specified by the STRING statement.

[LET] variable = expression
LET A=5
A$="HI"

Assigns the value of the expression on the right to the variable on the left. The keyword LET is optional.

**MEMEND expression
MEMEND 16R6800

This tells BASIC how much memory it can use. BASIC will not use any memory above this address. This is useful for obtaining memory for assembly language routines. Once MEMEND has been set, it is not reset except by executing another MEMEND statement, or by reloading of the BASIC system.

NEXT variable
NEXT J

This statement ends the FOR loop with the corresponding

variable.

**NOFLOW

Turns the flow trace off. See the FLOW statement.

ON expression GOSUB statement number list
ON J+2 GOSUB 100,600,620,110

This causes a GOSUB to the N-th statement number in the list, where N is the value of the expression. If there is no N-th statement number specified, then the next statement is executed. Upon execution of a RETURN statement in the subroutine, control returns to the statement following the ON ... GOSUB.

ON expression GOTO statement number list
ON J+2 GOTO 100,600,620,110

This goes to the N-th statement number in the list, where N is the value of the expression. If there is no N-th statement number then the next statement is executed.

**OUT(expression1) = expression2
OUT(2) = 10
OUT(16R02) = 16RFE

The value of the second expression is sent to the output port address specified by the first expression. Both expressions must be in the range 0 to 255. Hexadecimal values can be specified using the 16R form of integer constants, and octal values can be specified using the 8R form of integer constants.

Reading data from input ports is done using the IN function. The input operation returns a value which must be used in an expression or assigned to a variable, therefore it is treated as a function and is more fully described in the section on BASIC functions.

**POKE(expression1) = expression2
POKE(16R6FFF) = 2

This statement is used to store the value of the second

expression directly into the memory location specified by the first expression. The data value to be stored must be in the range from 0 to 255 (one byte of data), while the memory address must be in the range from 0 to 65535.

Reading directly from memory can be done using the PEEK function. Like the IN function, the PEEK function returns a value which must be used in an expression or assigned to a variable. Therefore PEEK is treated as a function and is further described in the section on BASIC functions.

PRINT expression list
PRINT "A=";A,"APPLES"

This displays the value of the expressions on the CRT. A comma is used to tab to the next field. Each field is 16 characters wide. A semicolon is used to print values next to each other. The TAB function can be used to tab to an absolute column. If the expression list is ended with a comma or semicolon, data displayed by later PRINT statements will appear on the same line. If not, data displayed by later PRINT statements will appear on new lines.

**SIZES(expression1,expression2,expression3)
SIZES(5,3,40)

Resets the number of memory bytes used for real numbers, integer numbers, and strings respectively.
"Expression1" is the number of bytes used for real numbers, "expression2" is the number of bytes used for integer numbers and "expression3" is the default length used for string variables. If the length of a string variable is explicitly declared in a DIM statement, the default size is not used. However, if a string variable is used without being specifically declared, then the default length from the SIZES statement is used.

The SIZES statement must be executed before any real or integer variables have been stored. If it is used, it should normally be the first statement of the BASIC program. For internal reasons, the size of integer values specified by expression2 must be less than the size of real values specified by expression1. The maximum size specification for real and integer numbers is 100 bytes, and the maximum length specification for

strings is 250 bytes. If no SIZES statement is executedthe default values are SIZES(5,3,40). The SIZES will be reset to these default values when execution begins using the RUN command.

STOP

The execution of a BASIC program is stopped by this statement. It may subsequently be resumed with the CONT statement.

**STRING expression
STRING ","

Sets the current string delimiter used in INPUT and GET statements. When strings are enclosed in quotes, the string delimiter is not involved. However, when strings are not enclosed in quotes, there must be some way to determine the end of the string. The end of an input string is delimited by either the end of the input record (the RETURN key for the INPUT statement) or the first occurence of the string delimiter. The string delimiter may only be one character long.

RANDOMIZE

This sets the random number generator to the current clock value.

READ variable list
READ J,K(1),A$

This reads values from the DATA statements and assigns them to the corresponding variables.

REM

Remark/comment line. This line is completely ignored by BASIC. A comment line can also be denoted by an exclamation character, "!", as the first character in the line.

RESTORE [statement number]

Restores the DATA statements so that subsequent READ statements start with the first DATA statement. The statement number parameter is not required, but if it is specified then subsequent READ statements will start with the DATA statement indicated by the statement number.

RETURN

This returns control to the statement following a GOSUB statement. See GOSUB.

RUN

Begins execution of a BASIC program. This command is an interactive command only and may not appear within a BASIC program.

5.0 OPERATORS

The following is a list of the operators implemented in MICROKIT BASIC. All of the operators are defined for numbers but only the comparison and concatenation operators are defined for strings.

Relational operators produce a logical "true" or "false" result. The value "true" is represented by the value one, while "false" is represented by the value zero. The values zero and one are also used by the IF statement to determine whether the test expression is true or false. Any expression which evaluates to a result of zero or one can be used as the test value in an IF statement.

Bit (Boolean) values are represented by integer numbers. Bit operators (AND, OR, NOT) require integer operands which are treated as 16-bit binary numbers. Integer numbers when used as bit values range from 0 to 65535 and represent the 16-bit binary equivalents. For example, the number 7 is treated as the binary number 0000000000000111 when used in bit operations, and 1033 is treated as 0000010000001001.

Logical operators work on a bit by bit basis. When the AND operation is performed on the two numbers above (7 and 1033) the result is 0000000000000001, the OR of the two numbers is 0000010000001111, and the NOT of the second number (1033) is 1111101111110110.

The list of operators is given from the lowest to highest precedence.

Binary (2 operand) Operators:

OR	Integer or function (5 OR 6 = 7)
AND	Integer and function (5 AND 6 = 4)
<>	Not equal to
=>	Equal to or greater than
=<	Equal to or less than
>=	Greater than or equal to
<=	Less than or equal to
>	Greater than
<	Less than
/=	Not equal to
=	Equal to
+	Addition, string concatenation
-	Subtraction
\	Integer division: x\y = INT(x/y)
*	Multiplication

/	Division
↑	Exponentiation

Unary (1 operand) Operators:

NOT	16 bit complement
+	No effect
-	Negation

6.0 FUNCTIONS

The following list of BASIC functions are described using the letters "n" and "s" to mean number and string arguments respectively. The symbol "=>" is used to indicate the result of the function examples.

6.1 String Functions:

```
CHAR(n)
CHAR(16R41) => "A"
```

Returns a character whose ASCII code is n.

```
ASC(s)
ASC("END") => 16R45 or 69
```

Returns the ASCII code of the first character in the string s.

```
LEFT(s,n)
LEFT("STRING",3) => "STR"
```

Returns the leftmost n characters of string s.

```
RIGHT(s,n)
RIGHT("STRING",3) => "ING"
```

Returns the rightmost n characters of string s.

```
MID(s,n1,n2)
MID("STRING",3,2) => "RI"
```

Returns the middle n2 characters of s starting at character n1.

```
LEN(s)
LEN("STRING") => 6
```

Returns the length of string s.

```
REPEAT(s,n)
REPEAT("A",5) => "AAAAA"
REPEAT("AB",5) => "ABABABABAB"
```

Returns a character string with s repeated n times.

INDEX(s1,s2)
INDEX("STRING","IN") => 4

Returns the position in s1 which is the first occurence of string s2 in s1. If s2 is not a substring of s1 then zero is returned.

VERIFY(s1,s2)
VERIFY("STRING","PQRST") => 4

Returns the first position in s1 of a character not found in s2. If all the characters in s2 are also in s1 then zero is returned.

STR(n)
STR(123.4) => "123.4"

Converts a number to a string.

FM(n,s)
FM(123.4,"$$$$,$$9V.99") => "$123.40"

Converts a number n into a string according to the format string s. Refer to the section on the Format function for further information.

VAL(s)
VAL("12.3E-2") => 0.123
VAL("12 + 13.5") => 25.5

Converts a string to a number. **The string can be any expression.

MIN(s1,s2)
MIN("ABC","ABD") => "ABC"

Returns the lesser of s1 and s2.

MAX(s1,s2)
MAX("ABC","ABD") => "ABD"

Returns the greater of s1 and s2.

6.2 Numeric Functions:

```
INT(n)
INT(123.4) => 123
INT(-123.4) => -124
```

Returns the greatest integer less than n.

```
FIX(n)
FIX(123.4) => 123
FIX(-123.4) => 123
```

Returns INT(ABS(n)).

```
FRAC(n)
FRAC(123.4) => 0.4
```

Returns the fractional part of n.

```
ABS(n)
ABS(123.4) => 123.4
ABS(-123.4) => 123.4
```

Returns the absolute value of n.

```
SGN(n)
SGN(123.4) => 1
SGN(-123.4) => -1
```

Returns either +1 or -1 depending on the sign of n.

```
MIN(n1,n2)
MIN(123.4,123.3) => 123.3
```

Returns the lesser of n1 and n2.

```
MAX(n1,n2)
MAX(123.4,123.3) => 123.4
```

Returns the greater of n1 and n2.

```
MOD(n1,n2)
MOD(150,12) => 6
```

Returns n1 mod n2 which is equal to n1-n2*INT(n1/n2).

RND(n)

Returns a random number. If n is non-zero, RND will use n as the seed otherwise an internal seed value will be used.

SQR(n)

Returns the square root of n.

LOG(n)

Returns the logarithm of n (base 10).

LN(n)

Returns the natural logarithm of n.

EXP(n)

Returns the exponential of n.

SIN(n)

Returns the sine of n.

COS(n)

Returns the cosine of n.

TAN(n)

Returns the tangent of n.

ATN(n)

Returns the arctangent of n.

6.3 Other functions:

CLOCK(0)

Returns a continually updated time, in seconds, which may be used for timing purposes. Timing is accomplished by sampling the clock at the start of an event, sampling the clock at the end of the event and subtracting the starting time from the ending time. The result is an elapsed time in seconds, accurate to 2 milliseconds.

The clock is disabled, however, during editing and when I/O to cassette tapes or to the EIA ports is in progress. If any of these operations are performed the clock will not be updated during the operation. The clock also ceases to function if an assembly language subroutine disables the processor interrupts or overlays the clock interrupt vector to handle the real-time clock directly.

IN(n)
IN(16E0) => data from input port E0 (hexadecimal)

Returns a number read from input port n. The port address n must be an integer value ranging from 0 and 255.

PEEK(n)
PEEK(16A23B) => data from memory location A23B (hexadecimal)

Returns the contents of memory location n. The memory address must be an integer value ranging from 0 to 65535.

PGMSIZE(0)

Returns the size of the program currently in memory, specified in bytes.

SPACELEFT(0)

Returns the unused space in memory, specified in bytes.

6.4 FM Function

For easy informal output no formatting is necessary. If fully formatted numbers are required, the FM function has been provided. This function has the capabilities of the "PRINT USING" of other BASIC systems, but since it is implemented as a function, it can be used with cassette tape and disk output as well as the usual printed output.

The FM function like any BASIC function can be used in an output statement, and it can also be used in statements where its result may be assigned to a string variable. Complicated line images are then assembled using the results of the FM function and the string functions and operators.

Essentially, the FM function converts numbers to strings. The first argument to FM is the number to be converted. The second argument gives a picture format description of the desired output. The second argument can be any expression evaluating to a string. Each character in the string (except a V) represents exactly one character in the output string. The characters listed below describe how certain characters are replaced with the appropriate digits of the number. Any characters not listed below are unchanged. If a number is too large then the entire string is filled with question marks.

To format a line of output, the FM function is invoked for each number to be placed in the line. The strings containing the formatted numbers are then combined using the concatenation operator "+" or the semicolon in the output statements (PRINT and PUT) to produce the final formatted line.

For example, consider the problem of producing a table with three columns of numbers formatted into dollars and cents. Three numbers would be printed on each line of output. A convenient format might be as follows:

```
A$ = "$$,$$$,$$9V.99"
```

This gives a 13 character result field with floating dollar sign and commas. The use of this format would produce the following kinds of results:

```
FM(1234.5,A$)        => "     $1,234.50"
FM(1234567.23,A$)    => "$1,234,567.23"
FM(.44,A$)           => "         $0.44"
```

Now, to format three numbers A, B, and C into a single line of output where each 13 character numeric field was separated by a blank, the following could be used:

```
PRINT "--- COL 1 ---<>--- COL 2 ---<>--- COL 3 ---"
PRINT FM(A,A$);" ";FM(B,A$);" ";FM(C,A$)
```

If the values for A, B, and C are respectively 1234.5, 12345.67, and 1234567.89, then the printed line would look as follows:

```
--- COL 1 ---<>--- COL 2 ---<>--- COL 3 ---

   $1,234.50      $12,345.67   $1,234,567.89
```

6.4.1 FM Picture Format Characters

Code	Description
9	Indicates a digit position of the number.
Z	Indicates a digit position. If this position is occupied by a leading zero it is replaced with a blank.
V	This is the only character which does not occupy a character space in the output string. It merely indicates where to align the decimal point. If no "V" occurs in the format string then it is assumed to be on the extreme right. Note that if a decimal point is to be printed it must be explicitly included in the format string. The "V" is provided to allow for scaling of the number in the formatting process. That is, a decimal point can be formatted into the output at a position which is different from the position of the decimal point in the actual number. The "V" indicates where in the output string the actual decimal point should be aligned.
$	Indicates a digit position. If more than one dollar sign is in the string then only the dollar sign closest to the leading digit is printed; the other dollar signs are replaced with blanks.
*	Indicates a digit position. If this position is to be replaced by a leading zero, the asterisk remains instead. When the digit position is indicated by a "9" then the leading zeros are replaced in the output string.
,	Commas appearing before the leading digit are replaced with either a blank, dollar sign or asterisk according to the context.

6.4.2 FM Function Examples

```
FM(23,"ZZ9")                    => " 23"
FM(-23,"ZZ9")                   => "-23"
FM(0,"ZZ9")                     => "  0"
FM(0,"ZZZ")                     => "   "
FM(2.3,"999")                   => "002"
FM(2.3,"ZVZZ")                  => "230"
FM(2.3,"AV.ZZ")                 => "2.30"
FM(2.3,"$,$$9V.99")             => "      $2.30"
FM(1234.5,"$$$,$$9V.99")        => " $1,234.50"
FM(9876,"***,**9V.99")          => "**9,876.00"
FM(.25,"***,**9V.99")           => "******0.25"
FM(123456,"999 999 99")         => "001 234 56"
FM(1.23,"BAL=$,$$9V.99")        => "BAL=    $1.23"
```

7.0 FILE INPUT/OUTPUT

MICROKIT Extended BASIC provides an extremely powerful file input and output system, with sophisticated data management capabilities. The BASIC input/output system can associate a file with any of the following devices:

- Floppy Disk Units
- Cassette Tape Units
- CRT Display
- System Keyboard
- Line Printer
- EIA Serial Ports
- Dummy Device

Full disk data management capabilities are provided by MICROKIT BASIC. Disk files can be opened, created, read or written randomly, renamed, and deleted. A long list of disk related functions provides for reading and changing file attributes, determining size of files and unused space on diskettes, repositioning read pointer, write pointers, and resetting end-of-files.

7.1 File I/O Features

The four primary BASIC statements for file I/O are OPEN, GET, PUT and CLOSE. The OPEN statement associates a file reference number with an I/O device, or if the device is disk, locates the file on diskette and prepares it for accessing. The GET statement is like the INPUT or READ statement except a file reference number is specified to indicate the source from which the data is to be obtained. The PUT statement is like the PRINT statement except a file reference number is again specified to indicate the destination device.

7.1.1 Record Based I/O

All input/output using the GET and PUT statements is on a record by record basis. For serial devices such as the keyboard, CRT display, line printer, and EIA serial ports, a record is a line of data terminated by a carriage-return character and on output, a line-feed character also. Thus output to these devices using PUT statements is on a line-by-line basis. Also input from these devices using GET statements is on a line-at-a-time basis.

I/O to cassette tape and floppy disk is also a record at a time.

Each GET and PUT statement reads or writes one record (sector on disk) every time it is executed.

7.1.2 Sequential and Random I/O

All devices except for floppy disk read or write sequentially. That is, each GET statement reads the next sequential record from the associated input device, and each PUT statement writes the next sequential record onto the associated output device. In the case of floppy disk both sequential and random accessing is possible. Any record in a floppy disk file can be accessed directly. However, floppy disk files can still be accessed in a sequential manner starting with the first record in the file.

7.1.3 Editor Compatible Files

There are two distinct types of files on tape and diskette. Files may be normal files or MICROKIT Editor compatible. Normal files on cassette tape are a series of records stored on tape, each record having a maximum length of 250 bytes.

Normal files on diskette are a collection of diskette records, each capable of storing a record with a maximum length 128 bytes. Normal disk files can be read or written sequentially or randomly.

Editor compatible tape and disk files have a special organization. These files are stored in a form that allows them to be read or written by the BASIC Editor, or by the MICROKIT system Editor. Editor compatible files may be read or written only in a sequential mode, and each GET or PUT references one sequential "line" which may be a maximum of 40 characters long. These "lines" correspond to the lines of data which are displayed by the Editor when the file is accessed by the Editor.

Editor compatible files are extemely efficiently formatted since data is compressed to eliminate blanks, and since multiple lines compressed into each record. Because of this compaction, and because multiple lines are stored in a single physical record on the media, it is not possible to randomly access the lines of a Editor compatible file.

Editor compatible files can be a very convenient way of entering data to be processed by BASIC programs. Data, both numeric and strings can be entered using the Editor and stored on disk or tape. This data can later be read an processed by BASIC programs.

It is also convenient to store data in Editor compatible format during debugging, because the data in the file can be easily

displayed for review using the Editor.

7.1.4 End-of-Page, End-file and Error Exits

All files have an ENDPAGE option which allows the programmer to specify some action to be performed every time a certain number of records have been outputted to the file. The number of records which causes the ENDPAGE exit is specified by the PAGESIZE parameter. While this option can be used with any file, it is most useful with output files destined for a printer. An EJECT statement is provided to make it easy to position the printer at the top of the next page.

Error and endfile exits can be specified for disk files and tape files. These parameters specify a statement number where control is transferred when an error or endfile is encountered.

7.1.5 Record Length Specification

As noted above all I/O using the BASIC data management facilities is performed on a record-by-record basis. Each GET or PUT transfers a unit of data to or from the associated I/O device. In some cases, a record length is required to accomplish the transfer. The GET and PUT statements provide for the length specification directly. However, a length can be associated with all GET and PUT statements performed with a particular file by specifing a global length in the OPEN statement.

The record length specification is optional on all types of I/O except to cassette tape. If no length is specified, the length of the data list to be written is used as the length for output, and the data read is assigned to the variables in the input list. For cassette tape output, the length specifies the length of the record to be written on tape. The output variable list is formatted into the record length specified. When reading cassette tape, the exact length of the records on tape must be specified in order to recover the data. Not all of the records on tape have to be the same length, however, the length of the record to be read must be known in order to be able to successfully read.

7.1.6 Data I/O Format

The data transferred by the GET and PUT statements is similar to the operation of the INPUT and PRINT statements. That is, the PUT statement formats one or more elements of a variable list into an output image for writing to the output device. The PUT statement may output a single string variable which corresponds to the

entire record, or it may output a list of numeric and string variables formatted just like in the PRINT statement.

Similarly, the GET statement works just like the INPUT statement. A single string variable may receive the data from the entire record read from the input device, or the record may be divided into a list of numeric and string variables. If numeric and string variables are interspersed in the data record, the string variables must be enclosed in quotes, as with the INPUT statement, or a string delimiter must be specified using the STRING statement. The STRING statement assigns a single character for use as the delimiter which denotes the end of non-quoted strings encountered in the input data. If this delimiter feature is not used, it is then necessary that quotes surrounding string variables be inserted into the output record on the PUT statement.

7.1.7 I/O to Disk

Sequential reading or writing of disk files is accomplished by simply executing a series of GET or PUT statements. Input and output to disk files can be intermixed without any explicit declaration on the OPEN statment. Intermixed input and output cannot be performed to sequential devices or to files specified as Editor compatible. Random accessing of disk files is accomplished when a record number is specified in the GET or PUT statements. This record number is a value between 1 and N where N is the number of records (sectors) in the file. The system automatically handles any situation where the tracks that make up the actual file on disk are not in a sequential, contiguous arrangement. And the system automatically makes all files appear to the BASIC programmer as if they were in fact a single contiguous block of records.

Pointers associated with sequential input and output to disk are not affected by random I/O. Consequently random I/O can be intermixed with sequential I/O. Sequential input or output resumes with the next record to be accessed when a GET or PUT without a record number specification is executed.

The sequential I/O pointers can be read and changed using a complete set of specialized functions.

When disk files are opened, the naming convention specifies whether the file is on drive 0 or 1, and whether the file exists or must be created. When a file is created in BASIC it is given an allocation of one track. However, when the track is filled using PUT statements, additional tracks will automatically be obtained. Disk file with larger than one track initial allocations can be obtained using the DOS System Monitor. Refer

to the "MICROKIT Disk Operating System Reference Manual" for further information about file capabilities.

7.2 File I/O Statements

In the following list of file I/O statements and functions, the expression "fn" refers to any expression evaluating to a number between 0 and 9. This number is the file reference number which associates various commands with the file specified by the OPEN statement. This file reference capability allows the programmer to reroute data to a different device merely by changing the OPEN statement parameters. For example output can be routed to the CRT display during initial test, because the speed of the CRT makes testing faster. After debugging is complete, output can then be routed to the line printer or to a printer connected via the serial port by simply changing the parameters on the file OPEN statement.

7.2.1 OPEN Statement

```
OPEN fn device parameters
OPEN 1 "*P"
OPEN 2 "1" EDIT
OPEN 2 "N:FILE-A" END 200
```

The expression "fn" specifies the file reference number to be associated with the opened file. "Device" is a string which specifies the I/O device and specific file-name if required. "Parameters" is a list of options, in any order, specifying specific requirements for the file. The "device" specification, and the "parameter" specifications are listed in the following sections.

7.2.1.1 OPEN Device Specification

The following lists the possible device specifications:

"" Dummy file. There is no actual I/O device associated with a dummy file. All output to this file is discarded. This designation can be used to inhibit the operation of all I/O statements referencing a specific file, without actually removing all of the I/O statements from the BASIC program.

"*" Specifies I/O to the CRT terminal. Output statements generate displays on the CRT and input statements obtain input from the keyboard.

"*P" Specifies output to a MICROKIT line printer attached to the system.

"*n" Specifies I/O to EIA serial port "n", where "n" is 1 or 2.

"n" Specifies I/O to cassette tape unit "n", when "n" is 1 or 2.

"filename" Specification of a disk file name implies I/O to diskette. The file name includes the specification of the disk drive number and whether the file exists on the diskette or must be created. If the file name is preceeded by a "N:" this means that the file is new and must be created. If the file name is preceeded by a "0:" or "1:", this specifies the disk drive that the file resides on. When neither drive specification is given, it is assumed that the file resides on drive 0. If both "N:" and "1:" must be specified, then the N: must be given first.

For further information on disk files names refer to the "MICROKIT Disk Operating System Reference Manual".

7.2.1.2 OPEN Parameters

EDIT

Specifies that the file is an Editor compatible source file. This can be used only with tape and disk files. In addition input and output cannot be mixed, and disk files cannot be accessed randomly.

CLEAR

Specifies that the disk file is to be cleared to an empty file. If CLEAR is not specified then sequential output is added at the end of the existing file. When CLEAR is specified, sequential output rewrites the file beginning with record one.

END statement number

Specifies the statement number to GOTO when an end-of-file condition is detected. This can only be specified for disk or tape files since these are the

only files for which an end-of-file can be detected by the system. If no END parameter is specified, and if an end-of-file is reached, a message will be displayed and program execution will be suspended.

ERROR statement number

Specifies the statement number to GOTO when an error condition is detected for disk or tape operations. When the disk error handling routine receives control, the type of error can be determined using the ERR and ERR$ functions. Refer to the description of these functions for further information. If no ERROR parameter is specified, if an error is detected, a message will be displayed and execution will be suspended.

ENDPAGE statement number

Specifies the statement number to GOSUB when the endpage condition occurs. An "end-of-page" condition occurs after the number of records specified by the PAGESIZE parameter have been outputted. For printed output, the "endpage" routine may be used to eject to the top of the next page, and print a title when required. While this feature is most useful for printed output, it can be used with any type of file. After the "endpage" subroutine is finished handling the condition, normal processing can continue with the statement after the PUT statement that generated the "endpage" using the RETURN statement.

PAGESIZE expression

Specifies the number of records to be transmitted before the ENDPAGE exit is executed. If no specification is made, the default value is 66. This is useful for printer output, because this corresponds to 11 inches on the line printer.

LENGTH expression

Specifies a record length value to be used with all GET and PUT statements unless specifically overridden in an individual GET or PUT statement.

XRATE expression

Specifies the data rate for EIA serial I/O as follows:

Code	Rate(bits/second)
0	50
1	75
2	110 (selects two stop bits for TTY)
3	134.5
4	150
5	300
6	600
7	900
8	1200

The default rate is 300.

NULLS expression

Specifies the number of null characters which are sent after the carriage-return and line-feed at the end of each record transmitted to the EIA lines. The default is 3. This provides a delay time to allow the carriage to return on serial character printers.

7.2.2 Other File I/O Statements

CLOSE fn

Closes file fn. This must be done when there are no further operations to be performed to the particular file. This statement is mandatory for output files to disk so that they are cleanly terminated, and can be properly reread. All tape files should be closed to provide for rewinding the tape.

GET fn [RECORD expression] [LENGTH expression] variable list
GET 1 A$
GET 2 A,B,C$,D,E$ RECORD A1

This reads a record from file fn and assigns the appropriate values to the variable list in the same manner as the INPUT statement. If a single string variable is specified in the list, data from the entire record is placed in the specified variable. If a list of variables are specified, values are assigned to the variables in the same way values are assigned in an INPUT statement. That is, numeric fields are delimited by at least one blank, while strings are delimited by being enclosed in quotes. If strings are not delimited by quotes, then a string variable will be assigned all the data remaining in the record, unless a string delimiter character has been chosen using the STRING statement. This allows one character to be reserved as the delimiter for the end of strings which are not enclosed in quotes.

For sequential devices such as cassette tape, the RECORD value has no significance. For direct access devices such as floppy disk, the RECORD parameter can be used to perform random accessing. If no value for RECORD is specified, then the GET statement automatically reads the next sequential record from the file. After the file has been opened using the OPEN statement, the first GET will begin reading the file starting at the first record.

Random access is accomplished by specifying a value for the RECORD parameter. The GET statement will then read the record specified. The value for RECORD is 1 for the first record in the file, and is n for the n-th record in the file.

The LENGTH parameter is only mandatory for tape input operations, and specifies the length of the record to be read. If all GET statements require the same length value, the length can be globally specified on the OPEN statement. Local specification on the GET statement allows for changing the length value for each read operation.

For devices other than tape, the length parameter is used to specify the maximum length of the data records to be transferred from the input device. For example, if only 100 bytes of data have been written into the fixed size 128 byte diskette sectors, the length parameter can be used to indicate that only the first 100 bytes of the record is valid data.

For disk input, each GET statement reads a disk sector, and any data not required for assignment to the variable list is ignored.

Both RECORD and LENGTH are optional parameters.

PUT fn [RECORD expression] [LENGTH expression] expression list
PUT 1 A$
PUT 2 A;B,C$,D;E$ RECORD A1

This statement writes a record to file fn. The expression list is evaluated and formatted in the same manner as a PRINT statement. If only one string variable is specified in the output list, the one variable then provides all of the data to be written into the record. If a variable list is specified, an output record image is formed just like in the PRINT statement, and the result is written to the output device.

For sequential devices such as cassette tape and line printers the RECORD parameter has no significance. However for randomly accessable devices such as floppy disks, this parameter can be used for direct accessing.

If the RECORD parameter is not specified for disk output, the records are written sequentially. If the file is opened with the CLEAR parameter specified, then the sequential output of records always begins with record 1. If this parameter is not specified, then sequential writing will add the new records after any records already exisiting in the file. When the RECORD

parameter is specified, then the PUT statement writes directly into the specified record. The specified record must be within the range of records already written. That is, a record cannot be written arbitrarily into a record which has never been written into before. Thus a file which is to be randomly accessed must be first "initialized" by the programmer by writing an initial set of data into the file.

The LENGTH parameter is mandatory on cassette tape files, but is optional to other devices. For tape files, if the length remains the same for all records to be written, it may be globally specified on the OPEN statement. If no length is specified, the length of the output image formatted from the output data will be used as the length of the record.

If the length of the output image is greater than the specified length, or the maximum length allowable by the output device, then the record will be truncated at the right, and a warning message will be displayed on the CRT.

Both RECORD and LENGTH are optional.

EJECT fn

Outputs blank lines until ENDPAGE is triggered. This allows a printer to be spaced up to the top of the next page.

All of the following statements are applicable to disk files only:

DELETE fn

Deletes disk file fn.

FREESPACE fn

Frees any unused tracks in disk file fn.

RENAME(fn)=expression

Renames disk file fn to the value of the string expression.

ATTRS(fn)=expression

Assigns new attributes to the disk file as follows:

1 --- Write Protect
2 --- Permanent File
4 --- Source File
8 --- Object File

Any combination may be assigned by summing the appropriate numbers.

EOF(fn)=expression

Resets the end-of-file for file fn to the record number specified by the value of "expression". That is, the record number specified by expression is designated as the last record in the file.

When a new file is created, the EOF is 0, and as new records are written to the file, the EOF is incremented. The current size of a file can be determined by the SIZE function, which returns the current EOF record.

The EOF statement can be used to reset the end-of-file so as to delete records from the file. This statement has no effect on the number of tracks allocated to the file on the disk.

GETSEEK(fn)=expression

Provides for resetting the pointer associated with sequential input from disk files. This can only be used on non-editor compatible files, and updates the pointer so that subsequent GET statements read starting from the record number specified by the value of the expression.

PUTSEEK(fn)=expression

Provides for resetting the pointer which is associated with sequential outputting to disk files. This can be used only on non-editor compatible files and updates the sequential pointer so that subsequent PUT

statements write into the record starting with the record number specified by the value of the expression. The sequential output pointer may only be set to a record number which is within the range of records that have previously been written into the file. That is, the value must be less than or equal to the value of the end-of-file.

LOAD fn

If fn is a disk or tape object file, it is loaded into memory. This statement is used for loading user written assembly language routines. If the program does not load into an unused area of memory, it can clobber portions of the BASIC system. Refer to the section covering assembly language subroutines for additional information.

REWIND expression

Rewinds tape cassette unit 1 or 2. The CLOSE statement for a cassette tape file will cause the tape unit to be rewound. However this statement is provided mainly as a direct command to allow the programmer to rewind the tape units when necessary. Never rewind the cassette tapes unless the BASIC system displays a message indicating that a rewind should be performed. If this rule is not followed, BASIC will not be able to maintain current information regarding the status of the tape drives, and the results may be unpredictable.

7.3 Disk File I/O Functions

All of the file I/O functions provided by MICROKIT BASIC are applicable only to floppy disk based files. The functions are used to return information about open files:

NAME(fn)

This is a string function which returns the name of the file opened with file reference number fn.

ATTR(fn)

Returns the attribute byte associated with file fn.

TRACKS(fn)

Returns the number of tracks used by disk file fn.

FREETRACKS(fn)

Returns the number of free tracks on the diskette that file fn resides.

SIZE(fn)

Returns the number of records in file fn. This value is equivalent to the number of the last record in the file, i.e. the end-of-file.

RECGET(fn)

Returns the value of the sequential input pointer indicating the last record to be read by a sequential GET statement.

RECPUT(fn)

Returns the value of the sequential output pointer indicating the last record to be written by a sequential PUT statement.

ERR(n)

Returns the completion code saved from the last disk I/O operation. It can be used to determine the type of error which caused the ERROR exit to be invoked. The value returned has the following meaning:

0 - Successful Operation
1 - Unrecoverable I/O Error
2 - End-File (if no end-file exit is specified)
3 - Disk Full
4 - File Not Found
5 - Duplicate Name
6 - Parmeter Error
7 - Drive Not Ready
8 - Permanent File (cannot be deleted)
9 - Write Protect (cannot be written into)

ERR$(n)

Returns a string which is an abbreviated message corresponding to the condition code returned by ERR.

8.0 ASSEMBLY LANGUAGE SUBROUTINES

This section briefly outlines how assembly language routines may be loaded and called from BASIC. At the end of this section an example is given.

Assembly language functions can be named in BASIC as a three letter variable. The first two letters must be "FA" and the last may be any letter. Thus up to 26 functions can be named. To define a function, one uses the DEF statement which tells BASIC the entry address of the function. After defining such a function it may be used in any expression followed by a parenthisized list of one to four arguments.

To reserve space the MEMEND statement is used to tell BASIC what the highest address it may use. The value of MEMEND may range from approximately 6800 (hexadecimal) to 440 (hexadecimal) less than the highest address in the system memory. This reserves all memory between the specified address and the highest value allowable for MEMEND (400 less than the highest machine address) for storage of assembly language functions. For the usual 32K system configuration, the area available for an assembly language program ranges from the value specified for MEMEND to 7BC0. If the assembly code can be confined to the memory area from C800 to DBC0 then the assembly language routine will be memory size independent. This is because addesses are always mapped into the highest order 8K block of memory according to the standard system memory configuring arrangement. Again, for a 32K system, the mapped addresses C800 to DBC0 correspond to the addresses 6800 to 7BC0.

To load an assembly language function into memory the LOAD statement is used. The LOAD statement loads a disk or tape object file generated by the MICROKIT Assembler into memory.

To access the arguments passed to an assembly language routine and to return a value, certain reserved locations are used as follows.

Address (hex)	Description
1B6	Pointer to argument 1. Two byte address stored in 8080 low/high format.
1B8	Pointer to argument 2.
1BA	Pointer to argument 3.
1BC	Pointer to argument 4.
1BE	Number of arguments passed by function call. One byte.
200	Return result. Area for returning result of

function. 256 bytes long.

All data items are stored in the format given in the section describing data types. The first byte of each data item specifies the type. Real numbers are X'01', integers are type X'02', and strings are type X'03'. Note that POKE and PEEK may also be used to pass information.

In the example below, an assembly language function displays on the CRT the string passed as the first argument. The assembly language routine was assembled and the object was put into the file "ASM.OBJ".

```
10! EXAMPLE OF CALLING AN ASSEMBLY LANGUAGE ROUTINE FROM BASIC
20 MEMEND !6R7000
30 OPEN !,"ASM.OBJ"
40 LOAD !
50 CLOSE !
60 DEF FAA = !6R7000
70 A = FAA("HI FOLKS")
80 ...
```

```
*****************************************
*                                       *
* EXAMPLE FUNCTION CALLED BY BASIC      *
*                                       *
* IF ONE STRING ARGUMENT IS PASSED      *
* THEN DISPLAY IT ON THE FIRST LINE     *
* OF THE CRT SCREEN                     *
*                                       *
*****************************************
        SPC
        ORG   X'7000'   ENTRY POINT
        LDA   NARGS     NUMBER OF ARGUMENTS PASSED
        CPI   1
        RNZ             RETURN ZERO
        LHLD  ARG1      FIRST ARGUMENT
        MOV   A,M
        CPI   3         STRING?
        RNZ             RETURN ZERO
        INX   H
        INX   H
        MOV   C,M       CURRENT LENGTH
        INR   C
        LXI   D,X'DC00' CRT SCREEN
LOOP    DCR   C
        RZ              RETURN IF DONE
```

```
        INX  H
        MOV  A,M        MOVE A CHARACTER AT A TIME
        STAX D          STORE
        INX  D
        JMP  LOOP
        SPC
NARGS   EQU  X'1BE'
ARG1    EQU  X'1B6'
        END  0
```

9.0 SAMPLE BASIC PROGRAM

The following sample program was chosen because it makes use of a number of the main features of the MICROKIT BASIC file input/output system.

The program reads Editor compatible source files from diskette or tape, and prints them on the line printer. This program is useful for listing BASIC programs stored on diskette or tape, as well as other data and programs stored in Editor compatible files.

```
    DIM U$(1),V$(1),W$(1)
    PRINT "LISTING PROGRAM FOR";
-   " BASIC PROGRAM FILES"
5   PRINT "ENTER FILE NAME OR";
-   " CASSETTE TAPE UNIT"
    INPUT B$
    W=0
    PRINT "DO YOU WANT TO DISPLAY IT";
    INPUT V$
    PRINT "DO YOU WANT IT PRINTED";
    INPUT U$
!
!   INITIALIZE FIRST PAGE
!   AND SET STRING DELIMITER
!
    OPEN 1 "*P" ENDPAGE 100
    J=0
    GOSUB 100
    STRING CHAR(0)
!
    OPEN 2 B$ EDIT ERROR 400 END 200
10  GET 2 A$
    IF U$="Y" PUT 1 A$
    IF V$="Y" PRINT LEFT(A$,39)
    GOTO 10
!
!   END PAGE ROUTINE
!
100 IF U$<>"Y" THEN RETURN
    PUT 1
    J=J+1
    PUT1:PUT1:PUT1
    PUT 1 "----"
    PUT1:PUT1
    PUT1 REPEAT(" ",60) "PAGE" J
    PUT1:PUT1
```

```
    RETURN
!
!   END ROUTINE
!
200 PRINT  "ALL DONE"
    IF U$="Y" THEN EJECT 1
    CLOSE2
    CLOSE1
    PRINT "ANY MORE FILES";
    INPUT W$
    IF W$="N" THEN STOP
    GOTO 5
!
!   ERROR ROUTINE
!
400 PRINT ERR$(0)
    CLOSE 1
    GOTO 5
    END
```

futuredata

11205 SOUTH LA CIENEGA BOULEVARD LOS ANGELES CALIFORNIA 90045 TWX 910-328-7202 TEL (213) 641-7700